SD211 TP Logistic Regression
WEI CHEN
weichen@whu.edu.cn

0 Import

```
import cervicalcancerutils as cancer
import numpy as np
from scipy.optimize import check_grad, line_search
import matplotlib.pyplot as plt
from pywt import threshold
```

```
filename = 'riskfactorscervicalcancer.csv'
X, y=cancer.load_cervical_cancer(filename)
n,p=X.shape
```

1. Tikhonov Regularisation

Question 1.1 Gradient, Hessian Matrix

$$f_1 = \frac{1}{n}\sum_{i=1}^{n} log(1 + exp(-y_i(x_i^T w + w_0))) + \frac{\rho}{2}\|w\|_2^2$$

Note $\overline{w} = \binom{w_0}{w}$, $\bar{x}_i^T = [1, x_i^T]$ and s $= [0,1, \dots ,1]$, then we can rewrite the objective function as

$$f_1 = \frac{1}{n}\sum_{i=1}^{n} log(1 + exp(-y_i\bar{x}_i^T\overline{w})) + \frac{\rho}{2}\|s \circ \overline{w}\|_2^2$$

$$\nabla_{\overline{w}} f_1 = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i\bar{x}_i^T\, exp(-y_i\bar{x}_i^T\overline{w})}{1 + exp(-y_i\bar{x}_i^T\overline{w})} + \rho(s \circ \overline{w})$$

And $f_1(w_0, w)$ is twice differentiable, and we calculate its Hessian matrix

$$Hessian = \nabla^2 f_1 = \frac{1}{n}\sum_{i=1}^{n} \frac{y_i^2\bar{x}_i\bar{x}_i^T\, exp(-y_i\bar{x}_i^T\overline{w})}{[1 + exp(-y_i\bar{x}_i^T\overline{w})]^2} + \rho I_s$$
$$= \frac{1}{n}A^T A + \rho I_s$$

where $A_i = \frac{y_i\, exp\left(-\frac{1}{2}y_i\bar{x}_i^T\overline{w}\right)}{1+exp(-y_i\bar{x}_i^T\overline{w})}\bar{x}_i$ and $I_s = diag(s)$

This matrix is definitely positive, so $f_1(w_0, w)$ is convex.

```
def tikhonov(X, y, w0w):
    val=0
    grad_w=0
    grad2_w=np.zeros((p+1,p+1))
    for i in range(n):
        x=np.insert(X[i,:],0,1).reshape(p+1,1)
        s=np.ones(p+1)
        s[0]=0
        Is=np.diag(s)
        z = np.exp(-y[i]*x.T.dot(w0w)[0])
        val=val+1/n*np.log(1+z)+1/n*rho/2*np.sum((s*w0w)**2)
        grad_w=grad_w+1/n*(-y[i]*x.T[0])*z/(1+z)+1/n*rho*(s*w0w)
        grad2_w=grad2_w+1/n*y[i]**2 * (x.dot(x.T)) * z / (1+z)**2 + 1/n*rho*
Is
    return val, grad_w, grad2_w
```

Question 1.2 check_grad

We want to verify the function defined above is correct, with the help of the function `scipy.optimize.check_grad`.

```
def f1(w0w):
    return tikhonov(X, y, w0w)[0]


def grad(w0w):
    return tikhonov(X, y, w0w)[1]
```

```
rho=1/n
w0w=np.random.rand(p+1)
print(check_grad(f1, grad, w0w, epsilon=1e-6))
```

This function will return the 2-norm of the difference between `grad(w0w)` and the finite difference approximation of *grad* using `f1`. The result we obtain is

`1.60841462877e-07`

This difference is small enough, which indicates the gradient of w calculated by function `grad` is verified.

Question 1.3 Newton Method

$$\boxed{\bar{w}^{(k+1)} = \bar{w}^{(k)} - \left(\nabla^2 f_1\left(\bar{w}^{(k)}\right)\right)^{-1} \nabla f_1(\bar{w}^{(k)})}$$

We fix the initial point as $\left(w_0^0, w^{(0)}\right) = 0$, and propose the stop condition as $\|\nabla f_1(\bar{w})\|_2 \leq 10^{-10}$.

```
def newton(w0w,epsilon):
    gradient=tikhonov(X, y, w0w)[1]
    G=list([])
    while np.sqrt(np.sum(gradient**2))>=epsilon:
        gamma=np.linalg.inv(tikhonov(X, y, w0w)[2])
        w0w=w0w-gamma.dot(gradient)
        gradient=tikhonov(X, y, w0w)[1]
        G.append(np.log10(np.sqrt(np.sum(gradient**2))))
    return w0w, G
```

```
epsilon=1e-10
rho=1/n
```

```
w0w=np.zeros(p+1)
w0w_opt,G=newton(w0w,epsilon)

plt.figure()
plt.plot(G,'--o')
plt.xlabel('iteration')
plt.ylabel('logarithmic 2-norm of grad_w')
plt.grid()
plt.show()
```

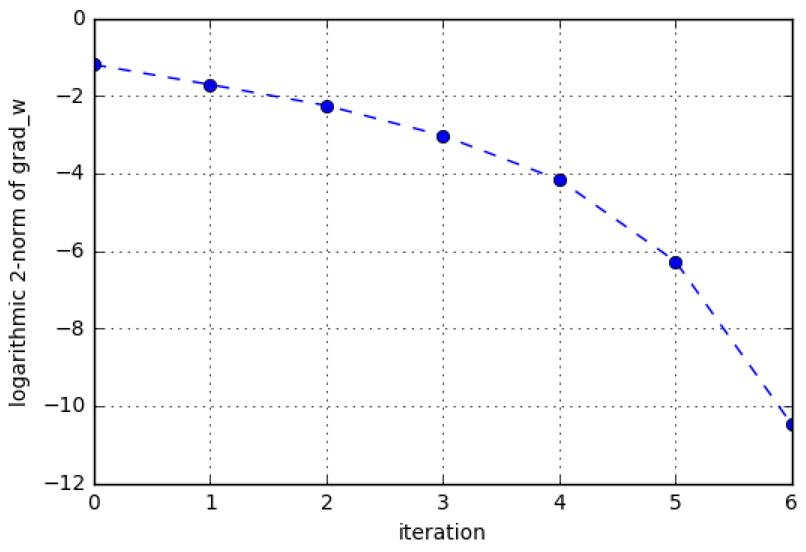

Figure. Evolution of $lg\|\nabla f_1(\overline{w})\|_2$

Question 1.4 Newton Method

If we take the initial point as $\left(w_0^0, w^{(0)}\right) = 0.3e$, we will find the algorithm doesn't converge.

Because one of the condition to ensure the convergence of Newton method is that the initial point should be chosen close enough to a local minimum x^*.

We verify that

$$\|(\overline{w}^* - 0)\|_2^2 = 2.39$$
$$\|(\overline{w}^* - 0.3e)\|_2^2 = 3.19$$

So we conclude that $0.3e$ is too far from the minimum $\overline{w}^*$, which doesn't ensure the convergence.

Question 1.5 Line Search

The idea of line search is to choose the pace γ_k adaptively using local information. To apply the line search to Newton method, we take

$$\boxed{\gamma_k = argmin_\gamma\, f_1(\overline{w}^{(k)} - \gamma\left(\nabla^2 f_1(\overline{w}^{(k)})\right)^{-1} \nabla f_1(\overline{w}^{(k)}))}$$

In Armijo's line search, we choose the pace length in each step as

$$\gamma_k = ba^l$$

where $a \in (0,1), b > 0$ and l is the smallest nonnegative integer such that

$$f_1\left(\overline{w}^+(ba^l)\right) \leq f_1\left(\overline{w}^{(k)}\right) + \beta \left\langle \left(\nabla^2 f_1\left(\overline{w}^{(k)}\right)\right)^{-1} \nabla f_1(\overline{w}^{(k)}), \overline{w}^+(ba^l) - \overline{w}^{(k)} \right\rangle$$

Basically, the larger β is, the larger γ will be; b determines the maximum pace length; a controls the number of levels of pace length. The following two figures show the roles of β and a.

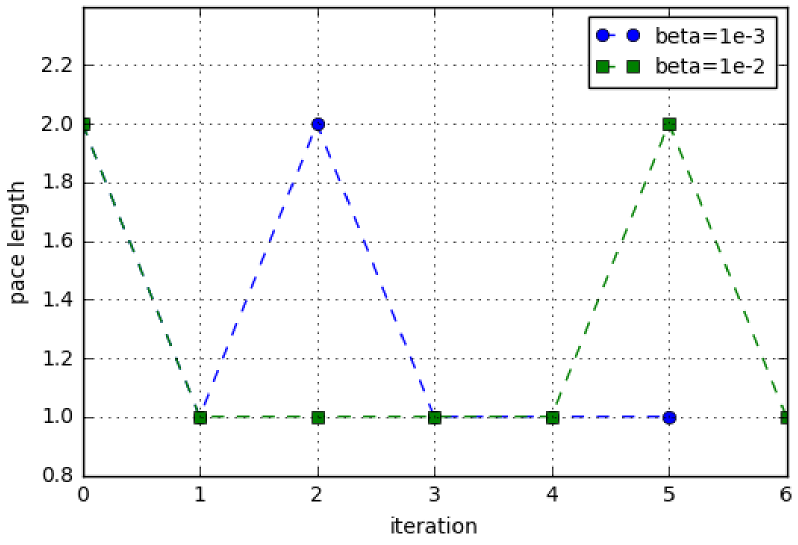

Figure. The larger β is, the more possible γ is selected as large value

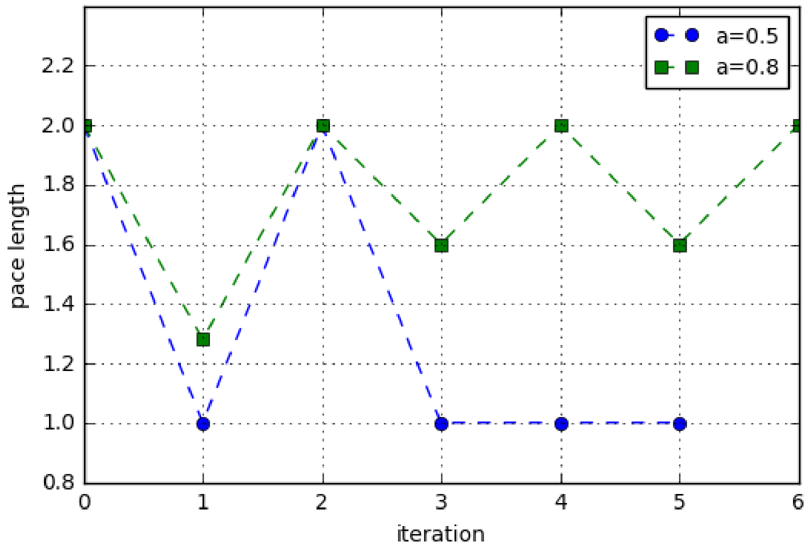

Figure. The smaller a is, the more possible γ is selected as large value

In order to let the algorithm converge as fast as possible, we select $\beta = 10^{-3}, b = 2, a = 0.5$.

```
def linesearch(f,grad,w0w):
    a=0.5
    b=2
    beta=1e-3
    l=0
    direction=np.linalg.inv(tikhonov(X, y, w0w)[2]).dot(tikhonov(X, y, w0w)
[1])
    w0wplus=w0w-b*a**l*direction
    while f(w0wplus)>f(w0w)+beta*direction.dot(w0wplus-w0w):
        l=l+1
        w0wplus=w0w-b*a**l*direction
    gamma=b*a**l
    return gamma
```

```
epsilon=1e-10
w0w=np.zeros(p+1)
gradient=grad(w0w)
G=list([])
while np.sqrt(np.sum(gradient**2))>=epsilon:
    gamma=linesearch(f1,grad,w0w)
    direction=np.linalg.inv(tikhonov(X, y, w0w)[2]).dot(gradient)
    w0w=w0w-gamma*direction
    gradient=grad(w0w)
    G.append(np.log10(np.sqrt(np.sum(gradient**2))))
w0w_opt=w0w


plt.figure()
plt.plot(G,'--o')
plt.xlabel('iteration')
plt.ylabel('logarithmic 2-norm of grad_w')
plt.grid()
plt.show()
```

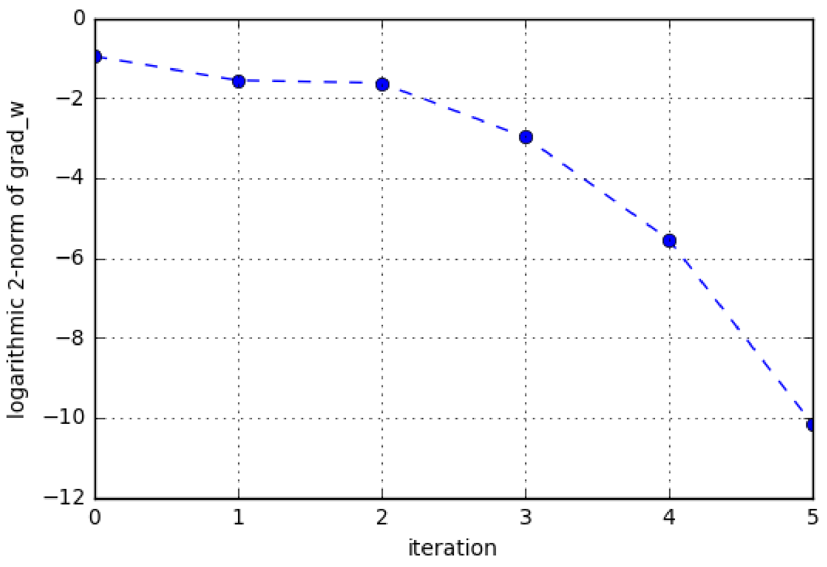

Figure. Evolution of $lg\|\nabla f_1(\overline{w})\|_2$ with $\beta = 10^{-3}, b = 2, a = 0.5, \epsilon = 10^{-10}$. $lg\|\nabla f_1(\overline{w})\|_2$ converges to 10^{-10} in 5 iterations.

2. Sparsity Regularisation

Question 2.1 Gradient, Hessian Matrix

$$F_2 = \frac{1}{n}\sum_{i=1}^{n} log(1 + exp(-y_i(x_i^T w + w_0))) + \rho\|w\|_1$$

$$\nabla_{\bar{w}} F_2 = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i\bar{x}_i^T \, exp(-y_i\bar{x}_i^T\bar{w})}{1 + exp(-y_i\bar{x}_i^T\bar{w})} + \rho \cdot sign(s \circ \bar{w})$$

where $sign(x) = \begin{cases} 1, \ x > 0 \\ [-1,1], \ x = 0 \\ -1, \ x < 0 \end{cases}$

Actually, F_2 is not differentiable at the point $\bar{w} = 0$, so we calculate its sub gradient instead.

Newton method requires that the objective function is three times continuously differentiable, however we can see that $\nabla_{\bar{w}} F_2$ is not differentiable at $\bar{w} = 0$. So Newton method cannot be applied to this case.

Question 2.2 Proximal Gradient Method

Rewrite the objective function as $F_2 = f_2 + g_2$

where

$$f_2 = \frac{1}{n}\sum_{i=1}^{n} log(1 + exp(-y_i x_i^T \bar{w}))$$

and

$$g_2 = \rho\|w\|_1$$

The proximal gradient method is given by

$$\boxed{w^{(k+1)} = prox_{\gamma_k, g_2}(w^{(k)} - \gamma_k \nabla f_2(w^{(k)}))}$$

Where $prox_{\gamma_k,g_2}$ is called **soft thresholding function** which is defined by

$$prox_{\gamma_k,g_2}(u)_i = \begin{cases} u_i - \rho\gamma_k, & u_i > \rho\gamma_k \\ 0, & -\rho\gamma_k \le u_i \le \rho\gamma_k \\ u_i + \rho\gamma_k, & u_i < \rho\gamma_k \end{cases}$$

and the gradient of f_2

$$\nabla_{\bar{w}} f_2 = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i\bar{x}_i^T \, exp(-y_i\bar{x}_i^T\bar{w})}{1 + exp(-y_i\bar{x}_i^T\bar{w})}$$

The Hessian matrix of F_2

$$\nabla_{\bar{w}}^2 F_2 = \frac{1}{n}\sum_{i=1}^{n} \frac{y_i^2\bar{x}_i\bar{x}_i^T \, exp(-y_i\bar{x}_i^T\bar{w})}{[1 + exp(-y_i\bar{x}_i^T\bar{w})]^2} = \frac{1}{n}A^T A$$

We have proved that this matrix is positive definite, so the objective function is convex.

```python
def sparse(X, y, w0w):
    val=0
    grad_w=0
    for i in range(n):
        x=np.insert(X[i,:],0,1).reshape(p+1,1)
        s=np.ones(p+1)
        s[0]=0
        z = np.exp(-y[i]*x.T.dot(w0w)[0])
        val=val+1/n*np.log(1+z)
        grad_w=grad_w+1/n*(-y[i]*x.T[0])*z/(1+z)
    return val, grad_w
```

Question 2.3 Proximal Gradient with Line Search

When the divergence between the objective values in two iterations is small enough, we assume the algorithm converges. And the principle of the algorithm is as follows:

```
Initialise
        $k = 0$,
        $w_0^{(0)}, w^{(0)}$,
        $\epsilon$
repeat
    • choose $\gamma_k = argmin_\gamma\ f_2(\overline{w}^{(k)} - \gamma \nabla f_2(\overline{w}^{(k)}))$
    • $w_0^{(k+1)} = w_0^{(k)} - \gamma_k \nabla f_2(w_0^{(k)})$
    • $w^{(k+1)} = prox_{\gamma_k, g_2}\left(w^{(k)} - \gamma_k \nabla f_2\left(w^{(k)}\right)\right)$
    • $k = k + 1$
until $\left|F_2\left(w_0^{(k+1)}, w^{(k+1)}\right) - F_2\left(w_0^{(k)}, w^{(k)}\right)\right| < \epsilon$
```

```python
def f2(w0w):
    return sparse(X, y, w0w)[0]


def F2(w0w):
    return sparse(X, y, w0w)[0]+rho*np.linalg.norm(w0w[1:],1)


def grad(w0w):
    return sparse(X, y, w0w)[1]
```

```python
def proximal(w0w,epsilon):
    gradient=grad(w0w)
    C=list([])
    k=0
    while k<2 or np.abs(C[-1]-C[-2])>=epsilon:
        gamma=line_search(f2, grad, w0w, -grad(w0w) )[0]
        w0w[0]=(w0w-gamma*gradient)[0]
        w0w[1:]=threshold(w0w-gamma*gradient,rho*gamma, mode='soft')[1:]
        gradient=grad(w0w)
        C.append(F2(w0w))
        k=k+1
  return w0w, C
```

```python
epsilon=1e-6
w0w=np.zeros(p+1)
rho=0.1
w0w_opt,C=proximal(w0w,epsilon)


plt.figure()
plt.plot(C,'--o')
plt.xlabel('iteration')
plt.ylabel('objective value')
plt.grid()
plt.show()
```

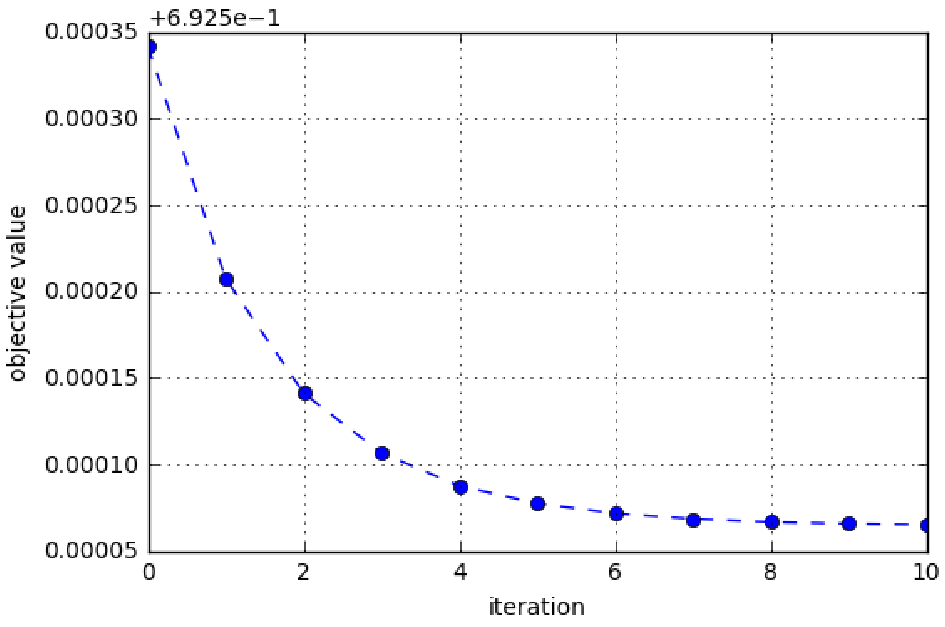

Figure. Evolution of $F_2(\overline{w})$ with $\epsilon = 10^{-6}$.

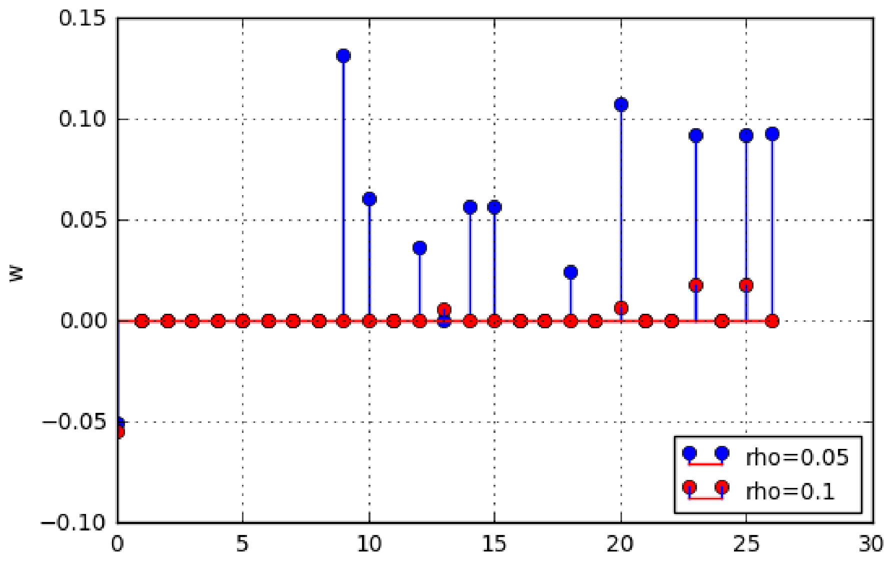

Figure. Solution of $\overline{w}$ with different sparsity level

We find that l_1-regularization will lead to sparse solution of $\overline{w}$. the sparsity level is determined by ρ. Larger ρ will lead to sparser $\overline{w}$.

3. Comparison

Question 3.1 Property of the two optimisation problems

- **Sparsity promotion**

The main difference between l_1 and l_2 regularization is that l_1 can yield sparse models while l_2 doesn't. Sparse model is a great property to have when dealing with high-dimensional data.

- **Solution of the optimization problem**
  l_2-regularized objective function is smooth. The solution that minimizes the objective function is the stationary point (0-derivative point).
  However, l_1 -regularized objective function is non-smooth. It's not differentiable at 0. The optimal solution of a function is either the point with 0-derivative or one of the irregularities (corners, kinks, etc.). So, it's possible that the optimal solution is 0 even if 0 isn't the stationary point.
- **Prediction error**
  Typically l_2 regularization is much better for minimizing prediction error rather than l_1 regularization. Because when two predictors are highly correlated, l_1 regularization will simply pick one of the two predictors. In contrast, the l_2 regularization will keep both of them and jointly shrink the corresponding coefficients a little bit.

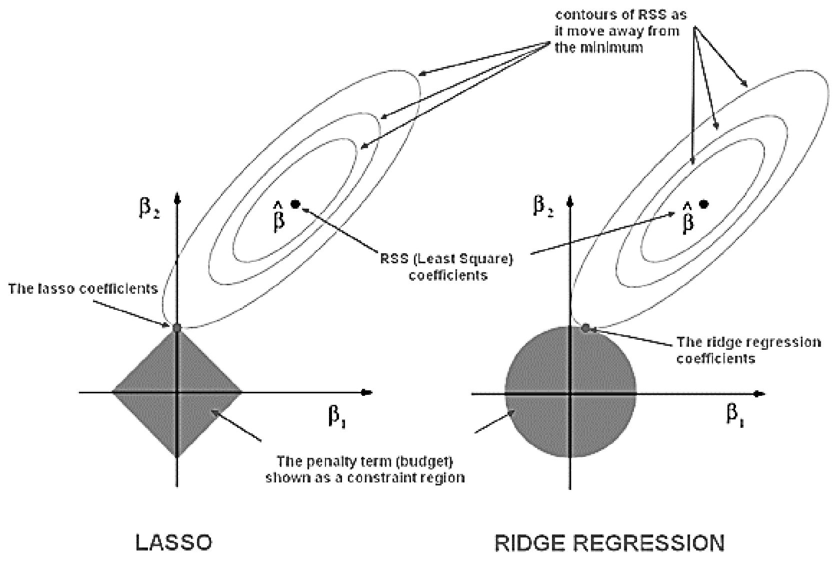
contours of RSS as it move away from the minimum
β_2
$\hat{\beta}$
RSS (Least Square) coefficients
The lasso coefficients
β_1
The penalty term (budget) shown as a constraint region
The ridge regression coefficients
LASSO
RIDGE REGRESSION

Figure. l_1 regularization leads to sparse solution, while l_2 regularization doesn't.

Question 3.2 Solution resulted from two types of regularisation

We fix the regularisation parameter ρ the same in two objective functions: $\boldsymbol{\rho = \frac{1}{n}}$.
Then we obtain the estimated $\overline{w}$ of two problems with different regularisations.

]:

```
plt.stem(r1,markerfmt='bo',label='Tikhonov regularisation')
plt.stem(r2,markerfmt='ro',label='Sparsity regularisation')
plt.grid()
plt.legend(loc=2,fontsize=10)
plt.ylabel('w')
plt.show()
```

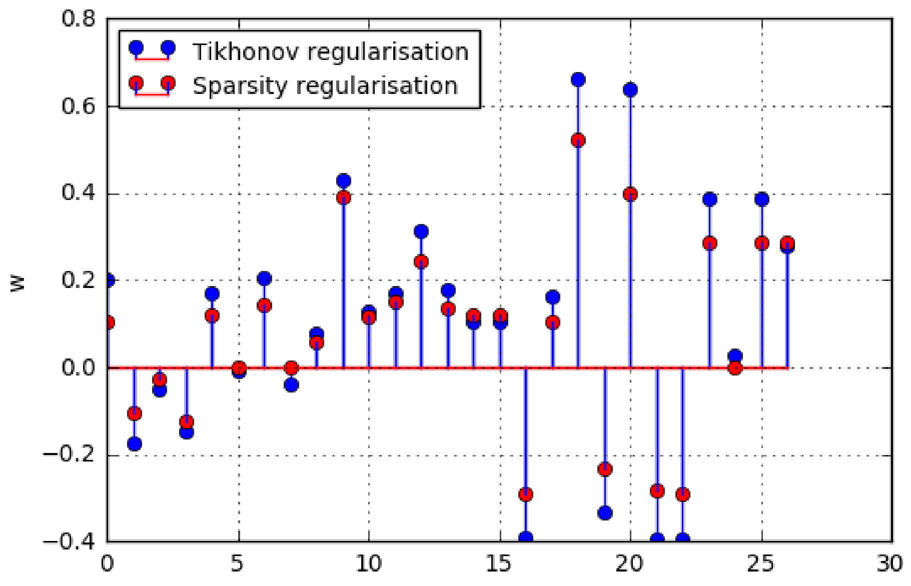

Figure. Solution of $\overline{w}$ of two optimisation problems

In logistic regression,

$$p(y_i = 1|x_i) = \frac{1}{1 + exp(-\bar{x}_i^T \overline{w})}$$

So we compute the probability based on $\overline{w}$ we estimated from the two problems.

```
y_est1=np.zeros(n)
y_est2=np.zeros(n)
for i in range(n):
    x=np.insert(X[i,:],0,1).reshape(p+1,1)
    y_est1[i]=1/(1+np.exp(-x.T.dot(r1)))
    y_est2[i]=1/(1+np.exp(-x.T.dot(r2)))
    y_est0=y*0.5+0.5
plt.figure()
plt.scatter(range(n),y_est0,c='black')
plt.scatter(range(n),y_est1,label='Tikhonov regularisation')
plt.scatter(range(n),y_est2,c='r',label='Sparsity regularisation')
plt.axhline(y=0.5,linestyle='--',color='black')
plt.grid()
plt.xlabel('i')
plt.ylabel('p(y=1|X,w)')
plt.legend(loc=1,fontsize=10)
plt.show()
```

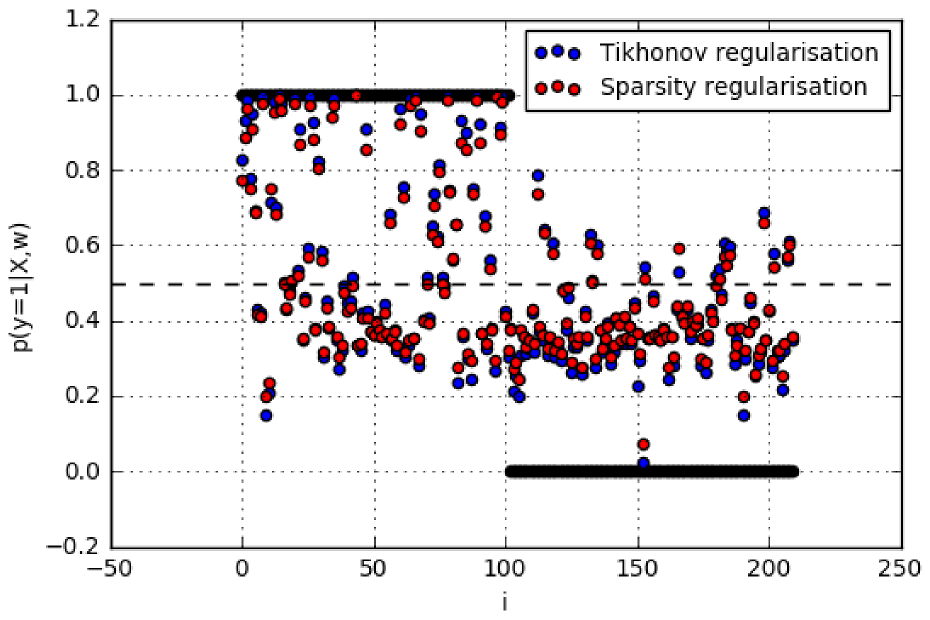

Figure. Estimated $p(y = 1|X)$ of two optimisation problems

We can conclude that both of this two regularizations will lead to a lot of prediction errors. Regarding this, it is very possible that the data set is not linear separable.